Software Engineering Institute

Managing Variation in Services in a Software Product Line Context

Sholom Cohen
Robert Krut

May 2010

TECHNICAL NOTE
CMU/SEI-2010-TN-007

Research, Technology, and System Solutions (RTSS) Program

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors would like to thank the participants of the 2007 and 2008 Workshops on Service-Oriented Architectures and Software Product Lines [SOAPL 2007, SOAPL 2008] held during the 2007 and 2008 Software Product Line Conferences. The work of these participants helped formulate the ideas presented in this report.

We would also like to thank Jörg Bartholdt, Bernd Franke, Christa Schwanninger, and Michael Stal for the example we used in Section 4 to illustrate our SOA concepts in a product line context: the health information exchange systems example.

Abstract

Software product line (SPL) and service-oriented architecture (SOA) approaches both enable an organization to reuse existing assets and capabilities rather than repeatedly redeveloping them for new systems. Organizations can capitalize on such reuse in software-reliant systems to achieve business goals such as productivity gains, decreased development costs, improved time to market, increased reliability, increased agility, and competitive advantage. Both approaches accommodate variation in the software that is being reused or the way in which it is employed. Meeting business goals through a product line or a set of service-oriented systems requires managing the variation of assets, including services. This report examines combining existing SOA and software product line approaches for variation management. This examination has two objectives: 1) for service-oriented systems development, to present an approach for managing variation by identifying and designing services explicitly targeted to multiple service-oriented systems, 2) for SPL systems, to present an approach for managing variation where services are a mechanism for variation within a product line or for expanding the product line scope.

1 Introduction

Software product line (SPL) and service-oriented architecture (SOA) approaches to software development share a common goal. They both enable an organization to reuse existing assets and capabilities rather than repeatedly redeveloping them for new systems. By adopting these approaches, organizations can capitalize on reuse in software-reliant systems to achieve business goals such as productivity gains, decreased development costs, improved time to market, greater agility, higher reliability, and competitive advantage [Cohen 2008].

We define systems that are built using these approaches as follows:

> An SPL is a set of software-intensive systems that share a common, managed set of features[1] that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [Clements 2001].

> ―SOA is a way of designing systems composed of services that are invoked in a standard way" [Lewis 2008b]. It is an architectural style or design paradigm that guides the composition and orchestration of services to implement business processes.

This report highlights the mutual benefits of combining systematic reuse approaches from product line development with flexible approaches for implementing business processes in an SOA. Under a combined SOA-SPL approach, developers build core assets, including services, and construct systems through the systematic reuse of these core assets in a predefined way. Product line approaches exploit the commonality across the products applying planned variation among their core assets. SOA provides a flexible mechanism for dealing with variation through services that are not bound to a specific product. In addition, a product line may be composed of service-oriented systems, if services are a basic element of composition and provide support for variation among members of the product family. Developers of service-oriented systems can apply product line approaches to extract service core assets from legacy systems and develop the architecture for new systems within the enterprise.

This report satisfies two objectives in examining the combining of existing SOA and SPL approaches:

1. for service-oriented systems development, to present an approach for managing variation to identify and design services targeted to multiple service-oriented systems
2. for SPL systems, to present an approach for managing variation where services are a mechanism for variation within a product line or for expanding the product line scope

The need for variation may be driven by market conditions, market opportunities, new technologies, and other business-related factors. This need may be stated in the form of business goals set by organizations that develop SPL or service-oriented systems. While goals are similar in both types of development organizations, they lead to different variation needs. For example, in service-oriented systems, the need generally includes reuse of a service across enterprise systems.

[1] A *feature* is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [Kang 1990].

For SPL systems, the need includes systematic reuse across a product line. At the same time, both types of systems must meet quality attributes that may vary across the breadth of SPL or service-oriented systems.

The next section of this report defines the contexts for variation management: SPL and service-oriented systems. Section 3 describes how SPL and SOA approaches can be combined for reuse. Section 4 discusses variation management in an SPL through use of services and the application of product line approaches to support SOA design decisions. Section 5 examines future work in this area.

2 The Basics of Software Product Lines and Service-Oriented Architectures

2.1 Software Product Lines

Software product line practice is the systematic use of core assets to assemble, instantiate, or generate the multiple products that constitute a software product line [Northrop 2009]. An organization's success in building a software product line lies in the ability of the organization to identify a group of like systems, the common and variant features of the systems, the assets used to develop the systems in the product line, and a plan for building the systems.

Product line scope and *product line analysis* define the boundaries and requirements of the software product line, based on the business goals of the organization [Chastek 2001]. The scope identifies the characteristics of the defined systems as being inside or outside the boundaries of the software product line. The scope also helps identify the common aspects of the systems, as well as expected ways in which they may vary.

Product line analysis applies established modeling techniques to engineer the common and varying requirements for a product line, based on input from stakeholders. Feature modeling within product line analysis captures the functional and nonfunctional requirements of the products in the product line and the decisions about common and variant capabilities and behaviors across the product line [Chastek 2001].

Composition elements of product line products are the product line *core assets*—artifacts or resources used in the production of products in an SPL. A core asset may be the architecture, a software component, a process model, a plan, a document, or any other artifact useful in building a system. Each core asset has an *attached process* that describes how the core asset is used in product production, including variant information. For example, the product line architecture must include mechanisms to support the explicitly allowed variations in the products within the product line scope; the attached process defines how that variation mechanism is used to generate the architecture for a specific product variant.

A *production plan* prescribes how the products are produced from the core assets and incorporates their attached processes. It includes the process to be used for building products (the *production process*) and the overall implementation approach (the *production method*) [Northrop 2009]. The production process is influenced by such contextual factors as *product constraints*[2] and *production constraints*[3] *and the production strategy.*[4]

[2] *Product constraints* are common and variant features and behavioral attributes associated with the products in the product line scope.

[3] *Production constraints* are any restriction on the timing, development environment, processes, or developer skills associated with the creation of products in a product line.

[4] The *production strategy* is the overall approach for realizing both the core assets and the products. It specifies the "prescribed manner" of development called for in the definition of a software product line.

2.2 Service-Oriented Architecture

A service-oriented architecture (SOA) provides a way to design, develop, deploy, and manage systems characterized by coarse-grained services that represent reusable business functionality and service consumers that compose applications or systems using the functionality provided by these services through standard interfaces [Lewis 2008a].

Two important principles of an SOA are the identification of services aligned with business drivers and the ability to address multiple execution environments by separating the service description (i.e., interface) from its implementation. Systems that employ an SOA architectural style are called service-oriented systems.

The major elements of service-oriented systems are services, service consumers, and an SOA infrastructure. The basic compositional elements of a service-oriented system are *services*—units of work that are applied to implement a business process to achieve a desired end result for a service consumer. They are self-contained, reusable components, configured to separate the standardized, well-defined service interfaces from the service's implementation. Service providers expose the capabilities of the service through the interface. Services are usually coarse-grained business functionality (such as customer lookup) rather than fine-grained functionality (such as customer address lookup). They can be globally distributed across organizations and reconfigured to support new business processes. A *service registry* is usually employed to enable service consumers to discover available services [Lewis 2008a].

Service consumers are clients for the functionality provided by the services. They query the registry for services with desired characteristics and compose or orchestrate applications using a service or composition of services. Some examples of service consumers are end-user applications, portals, intra-organizational enterprise systems and systems external to the organization, or even other services in the context of composite services [Lewis 2008a].

The SOA infrastructure is what connects service consumers to services. It usually implements a loosely coupled, synchronous or asynchronous, message-based communication model. The infrastructure often contains elements to support service discovery, security, data transformation, and other operations. One common SOA infrastructure is an enterprise service bus (ESB) to support SOA environments [Lewis 2008a].

3 Combining SPL and SOA Approaches

Meeting business goals through a product line or a set of service-oriented systems requires variation management of assets, including services. Variation management "comprises all activities to explicitly model, manage, and document those parts, which vary among the products of a product line" [John 2009]. Variation management applies to services in both the service-oriented and product line contexts. In both types of systems, variation points may be implemented either in a single service (where a service interface may offer parameterization or some other variation mechanism) or through similar services to address variation. To support variation mechanisms, SOA defines the infrastructure that selects and invokes the appropriate service and service variation for use in a specific system.

Examining the SOA and SPL approaches more closely—starting with the definitions provided in Sections 1 and 2—can help organizations identify and exploit the characteristics of each approach and combine them successfully for variation management. Those definitions deal with aspects of scope, the design approach, the source of variation, the application target, composition elements, and the technical approach. These aspects were the focus of discussion at the 2008 Workshop on Service-Oriented Architectures and Software Product Lines [SOAPL 2008]. The two definitions are decomposed in Table 1.

Table 1: Contrasting Aspects of SOA and Product Line Definitions

Aspects	Definition of an SOA	Definition of an SPL
Scope	*	A set of software-intensive systems
Design approach	A way of designing systems	**
Source of variation	*	that share a common, managed set of features
Application target	to implement business processes	satisfying the specific needs of a particular market segment or mission
Compositional elements	composed of services	and that are developed from a common set of core assets
Technical approach	invoked in a standard way	in a prescribed way

* not explicit in the definition
** not explicit in the definition but captured in the technical approach

SOA and SPLs address most of these aspects, and both define design approaches. SOA defines the composition and orchestration of services as an explicit design approach. An SPL defines the design approach in terms of the software architecture for that product line. SPLs also define a comprehensive set of core assets, including but not limited to a product line architecture, components, test cases, and so forth. In terms of targets, software product lines address a market segment or mission, while an SOA is intended to implement business processes within an enterprise. Finally, SOA prescribes a standard for service invocation, and software product lines require a prescribed way for constructing each product in the product line.[5]

[5] These differences are highlighted only to contrast the different emphasis within each definition. They are not meant to imply anything about the superiority of one over the other.

Further, significant distinctions between service-oriented and product line approaches emerge because of how scope and variation are utilized. A product line is built within an initial scope that usually evolves over time within a market segment due to technological or market changes and business decisions. Organizations may also identify ways in which to apply part or all of a core asset base to products in a new market. In an SOA approach, services are applied to address integration across systems in the enterprise and to capitalize on legacy investment [Butler 2006]. In an SOA approach, the scope grows or contracts to address user needs, and scoping, as practiced in software product lines, is generally not done.

While variation and variation management are key to an SPL approach, they are not highlights of an SOA approach. Yet, an SOA approach can be employed as a variation mechanism when core assets and products are developed in a product line. In an SPL approach, developers of software core assets would package desired capability as a service. That service may have built-in variation points that are accessible through parameterized service calls, or the service registry may identify variations among related service components that may include both the newly packaged capabilities and existing services from the enterprise. The SOA infrastructure provides an invocation mechanism to the needed service core asset for product development. Variation management in this context allows the tailored use of services to provide the exact, desired capability for a specific product. The dynamic nature of service invocation may support adaptation and more dynamic growth of a product line's scope. SOA may also support a more opportunistic response to changing market conditions with product line adaptation or new product lines.

For service-oriented systems, product line approaches could expand on the migration of legacy software to services, if multiple applications were seen as the targets for the newly exposed services. Variation management in this context helps identify the need for variation in services across a range of potential service-oriented systems. Similarly, the service registry could support the discovery of alternative services that may address variations among the applications that require those services. For service consumers, variation management helps identify a specific service that addresses an end-user need. This latter approach supports software product lines as well and provides software core assets with variation points for a specific product line product.

4 Variation Management for Services Under an SPL Approach

This section discusses variation management for services when an SPL approach is used and how the approach can be used to support SOA design decisions. An SPL approach can help organizations

- achieve specific SOA business goals—such as increased reuse and market responsiveness—through variation management of services. For example, developers of a comprehensive product line of medical information management systems can design a common patient authentication service with built-in variation points to support the management of healthcare, delivery of insurance coverage, retrieval of needed records, or scheduling of appointments.
- employ SOA service invocation as a variation mechanism. For example, the product line for a medical information management system may have different offerings for different users, such as hospitals, specialized clinics, medical offices, and individuals. The authentication service invocation protocol will support the variation needed for each type of offering to make the appropriate information available when requested and restrict unauthorized access.

To integrate SPL and SOA approaches, product line development organizations must apply variation management within SOA approaches, and service-oriented system developers must apply the approaches that product line developers use. This integration must provide variation management of services (e.g., variation points generated by the needs of different types of users of the medical information management system mentioned above) and must address services as a variation mechanism (e.g., the service invocation protocol that selects from among those variations).

The next four sections describe four principles of variation management in product line or service-oriented systems. To illustrate these principles, we use the example of health information exchange (HIE) systems. Bartholdt offers a meaningful example of SOA concepts in a product line context applied to HIE systems [Bartholdt 2008]. These systems provide distributed management of patient-related data for hospital chains, regions, or even whole nations. The case study illustrates the integration of SOA-based HIE implementations within a product line based on known product variation. Figure 1 [RTI 2007] illustrates some of the services obtained by the institutions supported by an HIE. Though not depicted here, an HIE may also obtain services from external sources.

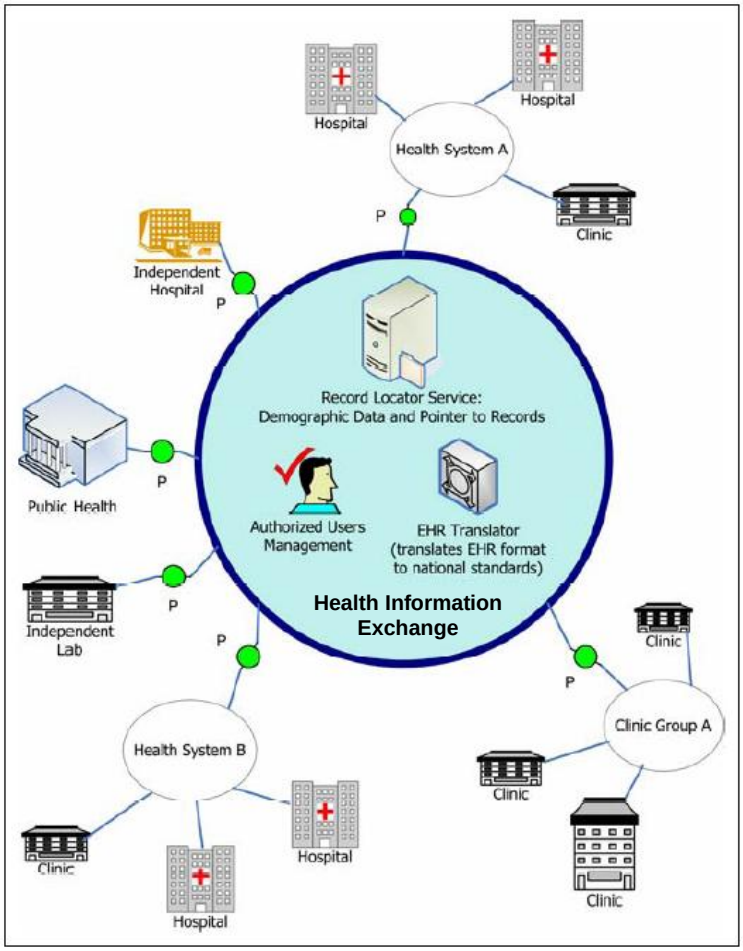

Figure 1: Product Line Capabilities in a Health Information Exchange

The four principles are listed below. Each principle is accompanied by an HIE system capability that can result from applying the principle.

1. Recognize the commonality and variants across the scope of a product line or across some group of service-oriented systems within the enterprise. *Product variation to support compartmentalized access to patient data depending on the user*
2. Leverage the recognized commonality by building core assets, including services, across the variants with established points of variation. *Core asset variation to support different health systems, clinics, hospitals, and labs*
3. Address the enterprise integration needs that service-oriented systems must offer. *Integration with external services*
4. Address end-user needs for variation within service-oriented systems. *User variation to accommodate unique work flow or user interface needs*

Section 5.1 discusses a final principle:

5. Establish production plans that use appropriate variation mechanisms to build product line or service-oriented systems based on these principles.

along with other future work for merging SPL and SOA practices.

Each of the following sections will also include a discussion of methods from the SOA-PL literature as they apply to one of the above four principles. The literature discussion will highlight potential contributions or areas for further investigation of methods to support the SOA-PL connection. The section will also discuss shortcomings in the SOA methods when applied directly to product line development. The HIE product line will illustrate how SOA methods can be enhanced to address product line needs in each of the four areas.

4.1 Recognize Commonality and Variants

Product line scoping identifies which potential products are in or out of the product line. Scoping supports this decision process by identifying what is common or varying among products in the product line. Product line members will share common features, while unique or alternative features documented in the product line scope will be a determinant in including or excluding a potential member. An understanding of commonality and bounding variation lets an organization determine whether a product can be built within the product line. The level of variation management across the product line is captured in the definition as a "common, managed set of features" and is essential to product line systems.

Methods that address reuse for service-oriented systems generally do not consider product variation among systems that will use or reuse those services. Developing service-oriented systems traditionally involves identifying services for a specified target but not defining a scope for a set of SOA-based systems implemented with those services. Some methods seek to identify services that legacy systems can expose, but with a specific application target for the service. The Carnegie Mellon® Software Engineering Institute developed such a method—the Service Migration and Reuse Technique (SMART) [Lewis 2008b]. SMART refers to the service identification as reuse—reuse of a legacy component or capability in the form of a service [Lewis 2008b]. New applications may also become a target for the service, and that would also be reuse. However, SMART does not address the need for a service (with some degree of variation) across a range of applications or a method for building in variation to accommodate multiple system targets.

Other methods also lack consideration of scope for application of a set of services. Balaji describes "services in an SOA … as reusable assets." Service assets in Balaji's method help leverage previous investments and may influence future investment with all the benefits of systematic reuse. Service assets support the quick construction of "new [business support] processes as a choreographed set of existing services" [Balaji 2007a]. The method includes the following among its listed SOA assets: business process models, reference architectures, and design patterns. However, Balaji's method does not define an initial scope for applying a service.

In applying product line techniques to SOA, users of the Component Based Development and Integration (CBDI) method look at both a suite of applications (CBDI concept of enterprise) and an application family (CBDI concept of product line) [Butler 2006]. Either may be the focus of the reuse of service assets. However, the CBDI method cautions users against trying to analyze how all the systems interact in a suite and doesn't address reuse across the family. Engineering services for systematic reuse would require this analysis as part of the scoping process.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

The HIE product line spans classes of customers across a diverse market. The product line must also accommodate data exchange standards. In Bartholdt's study, understanding product variation and scoping for HIE systems required the

- addressing of customer specifics or integration with existing third parties
- ability to quickly deploy a partial system or deploy an existing system to new platforms
- extensibility to address the emerging national data exchange standard

The scoping of product variation led to some specific variation points realized in service assets. The application of an SOA approach for the product line supported exposing master patient index (MPI) services to find duplicate patient data (Bartholdt 2008). However, this service is optional within the product line; some systems in the product line may use outside implementations for this service. In another example, workflow engine approaches within the product line applied chains of services to accomplish site-specific business processes.

As part of engineering services from legacy components or capability, the service migration can include a scoping process to identify existing or potential applications that could use the service. If these applications require slight variations in the service, the target scoping process can use product line approaches to

- identify service features that are common among the targets
- engineer variations or variant services to accommodate the target-unique features

For example, the patient authentication service must permit the retrieval of electronic data records for authorized users of a set of applications but will differ in terms of the kinds of information—diagnosis, billing, and prescribed treatments—that may be delivered for each class of user. The service may also vary by the types of identification data that must be entered to permit access—identification numbers, personal identifying questions, or biometrics. Engineering a common service could include provisions for these variations or offer multiple services to accommodate the needed variation.

4.2 Leverage Recognized Commonality

Product line scope identifies the need for variation across products. Software product line developers achieve variation across products by applying core assets with built-in variation points. To determine the coverage of assets within and across products, core asset development applies what has been termed *reuse scope*, that is, —which features [within product line scope] are to be developed for reuse, which for individual products without reuse" [Schmid 2005]. The intent of the reuse scope is similar to that of service migration from legacy within SMART—identifying which legacy capabilities should be exposed for reuse. The identified features may be implemented as requirements, within the architecture, or in other software-related core assets. Similarly, variations within features must be addressed within the appropriate core assets: requirements, architecture, components, test cases, or non-software core assets such as a configuration management plan.

Capturing variation is the responsibility of a core asset developer [Bachmann 2005]. In this context, the developer may be performing a requirements, architecture, testing, or technical management activity, depending on the core asset. Variation in the use of a core asset may be addressed by variation points, spelling out different uses of a core asset that include options and alternatives. Two other factors involved in the use of variation are variation mechanisms and the core asset

attached process. A variation mechanism defines the implementation approach that supports the application of a specific variation. For an architecture-level variation, a mechanism might include object-oriented extension through adding new attribute/method features. For a non-software core asset such as a document, an XML file may be interpreted to select alternative text paragraphs, graphics, or styling. The attached process gives the core asset user explicit guidance in using the core asset and its variation points and mechanisms.

The SOA approach may contribute to product line variation by supporting services as software core assets. In linking product line and service variation, a layered approach to service implementation demonstrates variation management within an SOA [Ludwig 2007]. This approach identifies customizable services at an atomic layer (e.g., purchasing service, human resources service). Service fulfillment (i.e., implementation) permits customization through the specifics of a request to an atomic service. The specific request "enables variability management by allowing the definition of variants of steps in the service fulfillment flow as alternative implementation flows." The layered service infrastructure provides the variation mechanisms for selection among service options and alternatives. For example, the service flow of a routine purchase request does not invoke the approval or multiple bid steps of a major purchase.

Other methods capture the goal of identifying variations within services for use across related systems or, potentially, in unforeseen ways. Feature analysis has been applied to service-oriented reengineering [Chen 2005] to support identifying services as core assets for application beyond the single target. Applying asset-based development approaches to services is also a focus of Balaji's work emphasizing tooling, life cycle, and governance [Balaji 2007a]. These three elements are also part of managing variability through the service registry [Balaji 2007b].[6] The use of an SOA for multiple applications may help identify variations that will lead to a change in product line scope or that can respond to new or changing market conditions. Using services to manage variability may lead to dynamic product line scope—where scope easily adapts to changing needs—or to the identification of totally new product lines based on parts of an existing core asset base.

The HIE system provides examples of both core asset variation and the use of service invocation as a variation mechanism. Service design can incorporate variation points [Bartholdt 2008]. Variations among in-scope HIE applications may lead to alternate service implementations, differentiations in workflows, or variability in customer environments (e.g., hospitals, clinics, or labs). To reduce the costs of managing multiple variation points, this approach has developed pre-configurations as a variation mechanism. The pre-configurations link workflow services or tasks for a particular class of user or situational need [Brandt 2003]. Use of pre-configurations addresses testability by reducing the number of service combinations.

[6] The service registry identifies web services available for discovery and invocation. The registry supports all variation management through interface descriptions such as the Web Service Description Language (WSDL) to match service providers to service users.

4.3 Address Enterprise Integration Needs

An SOA strategy addresses business goals such as the following that support enterprise integration [Lewis 2008a]:

- Integrate business partners (potentially across multiple computing platforms).
- Improve internal business processes through accessing legacy systems.
- Increase information to customers through customizable portals and information selection.

These goals apply across boundaries of existing product lines and require variations in product context to an extent not addressed in the traditional product line scope. In addition, the enterprise requires many different sets of systems, potentially product lines in their own right. Such integration variation—a context in which product line members must operate or interoperate—is outward focused. Traditional scope is more inward focused: What will the products in the product line do? An SOA that provides integration mechanisms supports the sharing of services across divergent systems in a context-independent fashion and leads again to dynamic product line scope.

How can product line and SOA concepts be applied in the enterprise context? An SOA may migrate to levels of increasing scope from project to line of business (LOB) to enterprise and beyond [Arsanjani 2005]. This work defines a Service Integration Maturity Model that is incremental and addresses key business goals such as quicker time to production, lower costs, and competitive differentiation. Initial states in service integration are ad hoc, with little reuse across service-oriented systems. As this service integration technology is adopted, an SOA may move through increasing levels of maturity. The first level is project focused, moving to LOB or product line focused, with a few common services. Still more common services depict the enterprise level, while value net adoption is characterized by common services and customizations. The value net level may achieve "dynamically reconfigurable services," which are context-aware, domain-specific services that operate by forming actively executable configurations on demand [Arsanjani 2006].

In the HIE environment, use of services as a variation mechanism supports enterprise integration. The varying needs across HIE systems are met by exchanging implementations with different variants or with implementations of external providers. The HIE uses this approach in MPI services to identify and track an individual patient and in a Document Registry/Document Repository. These capabilities may be implemented using internally developed or externally offered services from third parties. The SOA and product line connection has reduced integration time for connecting parts of the system to each other and to external systems.

4.4 Address End User Needs for Variation

Organizations building either product line or service-oriented systems may want those systems to offer end users the ability to perform their own customizations. That goal may be achieved through parameterization as a variation mechanism in product lines with static and varying parts under end-user control. Cummins engines and other automotive product lines apply end-of-line programming for users, such as engine idle speed [Clements 2001] or limiters for engine speed or revolutions per minute. Service-oriented systems often include customized portal interfaces that allow the user to choose the services and use them to make customizations.

Intelligence planning, collection, and analysis systems illustrate user variation within an SOA-based implementation of a product line [Jensen 2007]. Systems in the product line use a core asset base of services (called the VIPER Solution Framework) for situational awareness based on known product variation across the product scope. A single application may define user roles with individual systems tailored to specific user tasks. Variation mechanisms for this customization include both component selection and data behavior within the user interface. Individual users may perform their own customizations. Ease in training can be a business goal for the product line, through single-user interfaces for tasks involving integrated legacy components. For example, a user may customize one property editor for all integrated components instead of a different one for each component. This same property editor may be used for new components added to the system in the future.

In Batholdt's study, the use of product line approaches in HIE systems helped to identify features and an implementation of variability at the end-user level [Bartholdt 2008]. The focus on visible features from a business and domain perspective linked product development teams and the customer. Merging product line and SOA approaches supported flexibility in a dynamic business environment, enabling users to define unique workflow execution. In the HIE product line, each customer has a specific work environment and unique work processes. By exposing product line functionality as services, administrators at each customer site can perform workflow-engine-based composition of services using a variation mechanism to accommodate site-specific requirements. To support end-user variations, product line and SOA approaches have merged in creating core assets. The merged approach is not practiced widely within product lines, however. Service-oriented systems strive to achieve end-user variation and meet customization needs. Core asset development practices could achieve these goals by addressing variation management for end-user customization as part of a product line.

5 Future Work

We recommend focusing future work in three areas:

1. establishing production plans that use appropriate variation mechanisms to build product line or service-oriented systems. (This area was introduced initially in Section 4 as the last principle for successful application of variation management in services and their variation mechanisms.)
2. looking at SOA and SPL support for dynamic execution and dynamic variation mechanisms
3. investigating quality attributes as they relate to services and integrated service-oriented, product line applications

5.1 Establishing Production Plans to Encompass SOA

The principles discussed in Section 4 concentrate on product variation. A second category of variation is related to the production of products in a product line. Such variation can help an organization achieve goals for improving the ability to produce products, such as

- increased market share while maintaining current market leadership (i.e., standard for function, quality, and architecture)
- improved product marketability through improved productivity (lower costs, shorter time to market, and lower maintenance and support costs)
- ability to scale down for low-cost market as well as scale up
- ability to quickly deploy a partial system or deploy an existing system to new platforms
- distribution of work within and across development groups

These goals address production rather than product issues [Chastek 2009] and affect choices in scope, core asset development, and variation. Achieving quicker time to market may dictate the mining of service assets from a legacy system, thereby achieving variation by extracting two or more similar services. Increasing market share may require significant variation within a core asset in order to satisfy the many needs coming from the marketplace. A production strategy addresses these goals (see Section 2.1).

Incorporating an SOA approach within a production plan will affect the following:

- choices about service extraction and variation
- enterprise integration across systems using common services
- end-user-defined variation selections

SEI SMART-SYS, a variation of SMART [Lewis 2008b], addresses SOA-based systems development. This approach provides an understanding of a complete service-oriented system in the form of services, consumers, environment, risks, and costs. Currently under development, SMART-SYS considers many of the same issues that help shape the production strategy and, ultimately, the production plan. Merging SMART-SYS with product line production planning can help incorporate variation management with an SOA approach. This connection will be a focus of future work.

5.2 Support for Dynamic Service Execution

One frequently cited distinction between the SOA and product line approaches is the advantage of SOA in supporting dynamic execution [Cohen 2008]. This distinction generally focuses on dynamic service execution, while component execution is predefined. However, the product line architecture may also support a dynamic variation mechanism via plug-ins or some other "plug and play" architecture.

Another way to link dynamic product line approaches and an SOA is through system self-adaptation [Dolog 2008]. Self-adaptation supports access to variant information resources or functions as provided by services. Knowledge about the needs of a processing step provides appropriate information for service selection and configuration. These needs may vary based on the

- resource and information access environment
- application domain
- user/context
- configuration—variants and their meaningful combinations for specific purposes

Service providers offer many services that vary in functionality and quality attributes, while service consumers have similar requirements and need to achieve value through composing applications from reusable service assets. On the service provider side, there are service attributes in the form of features to describe the service. A variation point on the consumer side describes user needs via features. Under this approach, a matching service uses a feature model to match the features provided by a service to those needed by the consumer.

Future work in this area would need to consider, among other concerns, continuous configuration management. Dynamic service selection may deliver different services for the same service request, depending on service availability or other conditions. An error state or other trace must know the exact configuration that resulted from the specific service request. The use of a feature model as a dynamic variation mechanism is another concern.

5.3 Addressing Quality Attributes Through SOA and Product Line Approaches

A quality attribute is a property of a work product or goods by which its quality will be judged by some stakeholder or stakeholders. Quality attribute requirements such as those for performance, security, modifiability, reliability, and usability are key drivers in architecture decisions, including variation management [SEI 2009, Bass 2003]. Future work in this area will look at an actual service-oriented system to identify the appropriate services for multiple applications as targets. These services will be mined for shared service assets. The desired quality attributes of these applications will serve as drivers for design choices in the service-oriented product line or some hybrid applications.

The Factory pattern describes the entire software product line organization. The pattern includes steps to determine the scope of the product line, the production capability and its effective use, the parts from which products are built, and the process to monitor the operations and apply course corrections [Clements 2001]. This pattern could be applied to identify the scope of reuse across a potential product line of service-oriented systems and develop or reengineer existing services as software core assets. The effort would use appropriate architecture techniques to design the archi-

tecture for the product line and identify common or varying quality attributes [Clements 2002, Bianco 2007]. SEI Attribute-Driven Design [Bass 2003] or a similar technique would assure conformance to the identified quality attributes in developing services as core assets, and an architecture evaluation would use appropriate quality attributes to assess the fitness of the architecture. The production capability would apply services and other core assets as product parts for building products in the product line. Operations monitoring would assure the successful use of service mechanisms to achieve desired attributes for product development.

6 Conclusions

This report looks at combining existing SOA and SPL approaches for variation management. Both approaches

- encourage an organization to reuse existing assets and capabilities rather than repeatedly redevelop them for new systems
- enable organizations to capitalize on reuse to achieve similar business goals regarding software-reliant systems

Meeting business goals through a product line or through a set of service-oriented systems requires the variation of assets, including services, to be managed. This report emphasizes the importance of

- managing variation to identify and design services targeted to multiple service-oriented systems
- an approach for managing variation where services became a mechanism for variation within a product line or for extending product line scope

This report describes four principles leading to the successful application of variation management for both services and their use as a variation mechanism:

1. Recognize the commonality and variants across the scope of a product line or across some group of service-oriented systems within the enterprise.
   Following this principle helps organizations identify the services from legacy components or capabilities that could be used by existing or potential applications. For those applications requiring slight variations in the service, the scoping process could employ product line approaches to identify the service features that are common across all the products. Then, the organization could engineer variations or variant services to accommodate the products' unique features.
2. Leverage the recognized commonality by building core assets, including services, across the variants with established points of variation.
   Following this principle can lead organizations to address variation by packaging services as core assets with selected features. Service invocation can be used as a product line variation mechanism. An SOA approach may help identify variations that will lead to a change in product line scope or that can respond to new or changing market conditions. Using services to manage variability may lead to dynamic product line scope—where scope easily adapts to changing needs—or to the identification of new product lines based on parts of an existing core asset base.
3. Address the enterprise integration needs that service-oriented systems must address and that can be applied within product lines.
   Services used as variation mechanisms support enterprise integration. Varying needs across systems are met by implementations with different variants or with implementations that integrate with external services. The SOA and product line connection has caused reduced integration time for connecting parts of the system to each other and to external systems. In this context, enterprise integration could encompass many different sets of systems and

product lines, enabling SOA integration mechanisms to share services across divergent systems in a context-independent fashion and lead again to dynamic product line scope.

4. Address end-user needs for variation within service-oriented systems that can be applied within product lines.
   Product line approaches can identify features and an implementation of variability at the end-user level. By merging product line and SOA approaches, product functionality can be exposed as services, enabling the workflow-based composition of services using a variation mechanism to accommodate site-specific requirements. Merging these approaches with core asset development practices could elevate variation management for end-user customization needs in a product line.

Future work should capture promising techniques for applying these results to service identification methods such as SMART and to variation mechanisms that apply an SOA within a product line. A future workshop may be built around such techniques and bring together researchers and practitioners whose work has been documented in this report.

References

URLs are valid as of the publication date of this document.

[Arsanjani 2005]
Arsanjani, Ali & Holley, Kerrie. *Increase Flexibility with the Service Integration Maturity Model (SIMM)*. http://www.ibm.com/developerworks/webservices/library/ws-soa-simm/ (2005).

[Arsanjani 2006]
Arsanjani, Ali & Diaz, Jorge. *Service Integration Maturity Model (SIMM): Introduction*. http://www.opengroup.org/conference-live/uploads/40/11051/3_Diaz.pdf (2006).

[Bachmann 2005]
Bachmann, F. & Clements, P. *Variability in Software Product Lines* (CMU/SEI-2005-TR-012, ADA450337). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/library/abstracts/reports/05tr012.cfm

[Balaji 2007a]
Balaji, Rishi S. *Apply Asset-Based Development to Services in an SOA, Part 1: SOA and Asset Development Tooling, Life Cycle, and Governance*. http://www.ibm.com/developerworks/webservices/library/ws-soa-asset1/?S_TACT=105AGX04&S_CMP=EDU (2007).

[Balaji 2007b]
Balaji, Rishi S. *Apply Asset-Based Development to Services in an SOA, Part 2: Manage and Monitor Service Assets and Metadata*. http://www.ibm.com/developerworks/webservices/library/ws-soa-asset2/index.html?S_TACT=105AGX04&S_CMP=EDU (2007).

[Bartholdt 2008]
Bartholdt, Jörg, Franke, Bernd, Schwanninger, Christa, & Stal, Michael. "Combining Product Line Engineering and Service-Oriented Architecture in Health Care Infrastructure Systems: Experience Report," 115-122. *Proceedings of the 12th International Software Product Line Conference, Second Volume*. Limerick, Ireland, September 2008. Lero International Science Centre, 2008 (ISBN 978-1-905952-06-9).

[Bass 2003]
Bass, Len, Clements, Paul, & Kazman, Rick. *Software Architecture in Practice*, *2nd ed*. Addison-Wesley, 2003. http://www.sei.cmu.edu/library/abstracts/books/0321154959.cfm

[Bianco 2007]
Bianco, Phil, Kotermanski, Rick, & Merson, Paulo. *Evaluating a Service-Oriented Architecture* (CMU/SEI-2007-TR-015). Software Engineering Institute, Carnegie Mellon University, 2007. http://www.sei.cmu.edu/library/abstracts/reports/07tr015.cfm

[Brandt 2003]
Brandt, Samuel & Dehaan, Jan. *A System and User Interface Supporting Task Schedule Configuration*. http://www.wipo.int/pctdb/en/wo.jsp?IA=US2002026970&DISPLAY=DESC (2003).

[Butler 2006]
Butler, John. "Applying Product Line Techniques to SOA." *CBDi Journal* (February 2006): 22-31. CBDI-Everware, 2006.

[Chastek 2001]
Chastek, Gary, Donohoe, Patrick, Kang, Kyo Chul, & Thiel, Steffen. *Product Line Analysis: A Practical Introduction* (CMU/SEI-2001-TR-001, ADA396137). Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/library/abstracts/reports/01tr001.cfm

[Chastek 2009]
Chastek, Gary J. & McGregor, John D. "Modeling Variation in Production Planning Artifacts," 45-50. *Proceedings of the Third International Workshop on Variability Modelling of Software-intensive Systems*. Sevilla, Spain, January 2009. ICB-Research Report No. 29, 2009. http://www.vamos-workshop.net/proceedings/VaMoS_2009_Proceedings.pdf

[Chen 2005]
Chen, Feng, Li, Shaoyun, Yang, Hongji, Wang, Ching-Huey, & Chu, William Cheng-Chung. "Feature Analysis for Service-Oriented Re-Engineering," 201-208. *Proceedings of the 12th Asia-Pacific Software Engineering Conference*. Taipei, Taiwan, December 2005. IEEE Computer Society, 2005.

[Clements 2002]
Clements, Paul, Kazman, Rick, & Klein, Mark. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.
http://www.sei.cmu.edu/library/abstracts/books/020170482X.cfm

[Clements 2001]
Clements, P. & Northrop, L. M. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. http://www.sei.cmu.edu/library/abstracts/books/0201703327.cfm

[Cohen 2008]
Cohen, Sholom & Krut, Robert. *Proceedings of the First Workshop on Service-Oriented Architectures and Product Lines* (CMU/SEI-2008-SR-006). Software Engineering Institute, Carnegie Mellon University, 2008. http://www.sei.cmu.edu/library/abstracts/reports/08sr006.cfm

[Dolog 2008]
Dolog, Peter & Schafer, Michael. *Feature Based Design of Web Service Transaction Compensations*. *Proceedings of the 12th International Software Product Line Conference*. Limerick, Ireland, September 2008. IEEE Computer Society, 2008. http://splc.net/prev-conferences/soapl-2008.pdf

[Jensen 2007]
Jensen, Paul. "Experiences with Product Line Development of Multi-Discipline Analysis Software at Overwatch Textron Systems," 35-43. *Proceeding of the 11th International Software*

Product Line Conference. Kyoto, Japan, September 2007. IEEE Computer Society, 2007 http://splc.net/prev-conferences/soapl-2007.pdf

[John 2009]
John, I. & Pech, D. "Scalable Variability Instantiation Strategies." Scalable Modeling Techniques for Software Product Lines (SCALE 2009) Workshop, *13th International Software Product Line Conference (SPLC 2009).* San Francisco, California, August 2009. http://www.splc.net/prev-conferences/splc-2009.pdf

[Kang 1990]
Kang, K., Cohen, S., Hess, J., Novak, W., & Peterson, A. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-021, ADA235785). Software Engineering Institute, Carnegie Mellon University, 1990. http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm

[Lewis 2008a]
Lewis, Grace A. & Smith, Dennis B. *Service-Oriented Architecture and its Implications for Software Maintenance and Evolution*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4659243&isnumber=4659234 (2008).

[Lewis 2008b]
Lewis, Grace A., Morris, Edwin J., Smith, Dennis B., & Simanta, Soumya. *SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture Environment* (CMU/SEI-2008-TN-008). Software Engineering Institute, Carnegie Mellon University, 2008. http://www.sei.cmu.edu/library/abstracts/reports/08tn008.cfm

[Ludwig 2007]
Ludwig, Heiko, Bhattacharya, Kamal, & Setzer, Thomas. "A Layered Service Process Model for Managing Variation and Change in Service Provider Operations," 484-492. *Proceedings of Web Information Systems Engineering – WISE 2007* (Lecture Notes in Computer Science volume 4831). Nancy, France, December 2007. Springer, 2007.

[Northrop 2009]
Northrop, L. & Clements, P. *A Framework for Software Product Line Practice, Version 5.0.* http://www.sei.cmu.edu/productlines/tools/framework/ (2009).

[RTI 2007]
Research Triangle Institute. *Interim Report on Solutions to Barriers to the Electronic Exchange of Health Information,* 2007. http://www.health.state.mn.us/e-health/mpsp/mpspsolrpt011707.pdf

[Schmid 2005]
Schmid, Klaus & Biffl, Stefan. "Systematic Management of Software Product Lines." *Software Process Improvement and Practice (Special Issue on Software Variability: Process and Management) 1,* 10: 61–76, 2005. Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spip.215.

[SEI 2009]
Software Engineering Institute. *Software Architecture Glossary*. http://www.sei.cmu.edu/architecture/start/glossary/ (2009).

[SOAPL 2007]
—Wdkrshop on Service-Oriented Architectures and Software Product Lines - What is the Connection (SOAPL 2007).” *Proceedings of the 11th International Software Product Line Conference*. Kyoto, Japan, September 2007. http://splc.net/prev-conferences/soapl-2007.pdf

[SOAPL 2008]
—Wdkrshop on Service-Oriented Architectures and Software Product Lines - Putting Both Together (SOAPL 2008).” *Proceedings of the 12th International Software Product Line Conference*. Limerick, Ireland, September 2008. http://splc.net/prev-conferences/soapl-2008.pdf

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	May 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Managing Variation in Services in a Software Product Line Context	FA8721-05-C-0003

6. AUTHOR(S)

Sholom Cohen and Robert Krut

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TN-007

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (maximum 200 words)

Software product line (SPL) and service-oriented architecture (SOA) approaches both enable an organization to reuse existing assets and capabilities rather than repeatedly redeveloping them for new systems. Organizations can capitalize on such reuse in software-reliant systems to achieve business goals such as productivity gains, decreased development costs, improved time to market, increased reliability, increased agility, and competitive advantage. Both approaches accommodate variation in the software that is being reused or the way in which it is employed. Meeting business goals through a product line or a set of service-oriented systems requires managing the variation of assets, including services. This report examines combining existing SOA and software product line approaches for variation management. This examination has two objectives: 1) for service-oriented systems development, to present an approach for managing variation by identifying and designing services explicitly targeted to multiple service-oriented systems, 2) for SPL systems, to present an approach for managing variation where services are a mechanism for variation within a product line or for expanding the product line scope.

14. SUBJECT TERMS	15. NUMBER OF PAGES
variation management, software product line, SPL, service-oriented architecture, SOA	32

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL